



ML 06: PRACTICAL ADVICES

지난시간엔 *back propagation* 구현해 보고 여기에 적용할 수 있는 소소한 것들 *random initialization* 과 *gradient checking* 등도 알아 보았다.

머신러닝을 단순히 아는것과, 실전에서 사용할 수 있다는 건 큰 차이가 있다. 이번 시간에는 실전에서 필요한 여러가지 팁들에 대해 설명한다. 후반부에서는 스팸 분류기를 통해 간단한 머신러닝 시스템을 설계해 본다.

Diagnostics

[http://chart.apis.google.com/chart?cht=tx&chl=J\(%5Ctheta\)%20%3D%20%7B1%20%5Cover%202m%7D%20%5B%5Csum_%7Bi%3D1%7D%5Em%20\(h_%5Ctheta\(x%5E%7B\(i\)%7D%20-%20y%5E%7B\(i\)%7D\)%5E2%20%2B%20%5Clambda%5Csum_%7Bj%3D1%7D%5Em%20%5C%20%5Ctheta_j%5E2%5D](http://chart.apis.google.com/chart?cht=tx&chl=J(%5Ctheta)%20%3D%20%7B1%20%5Cover%202m%7D%20%5B%5Csum_%7Bi%3D1%7D%5Em%20(h_%5Ctheta(x%5E%7B(i)%7D%20-%20y%5E%7B(i)%7D)%5E2%20%2B%20%5Clambda%5Csum_%7Bj%3D1%7D%5Em%20%5C%20%5Ctheta_j%5E2%5D)

집 가격에 대한 *linear regression* 가설을 세웠는데 *error* 가 좀 큰 것을 발견했다. 어떻게 해야할까?

- (1) Get more training examples
- (2) Try smaller sets of features
- (3) Try getting additional features
- (4) Try adding polynomial features
- (5) Try decreasing *lambda*
- (5) Try increasing *lambda*

이중에서 더 많은 트레이닝 셋을 추가하는 것은 별로 도움이 안 될 수도 있다. (이유는 뒷 부분에서 논의한다.)

알고리즘의 정상 동작여부를 파악할 수 있는 몇 가지 판별법을 알아보자. *gradient checking* 이 그랬던 것처럼, 구현하는데는 좀 시간이 걸려도 디버깅에 드는 시간을 많이 줄여준다.

A diagnostic can sometimes rule out certain courses of action (changes to your learning algorithm) as being unlikely to improve its performance significantly

Evaluating a hypothesis

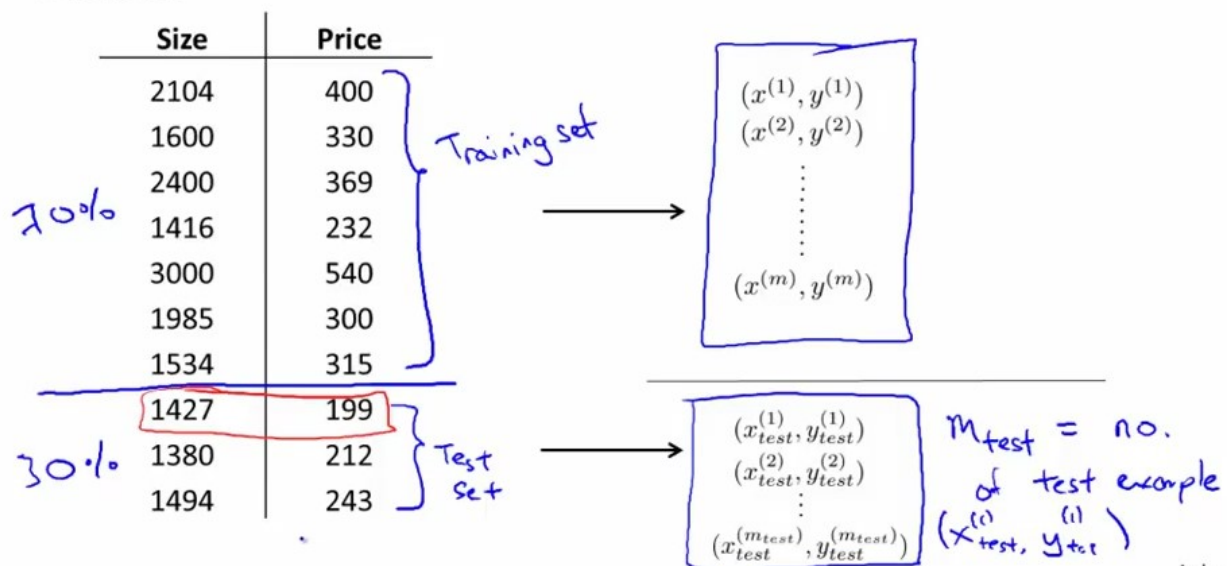
low training error 를 갖는 *hypothesis* (가설) 이 항상 좋은 건 아니다. *overfitting* 이 발생할 수 있기 때문이다.

그리고 *feature* 가 많을 수록 *plotting* 하기 힘들기 가설을 평가할 다른 방법을 찾아야 한다. 단순히 그리는 것 만으로 모든 가설을 평가하긴 어렵다.

한가지 평가 방법으로 전체 *training set* 을 70% / 30% 로 분리해 70% 은 *training set* 으로 나머지 30% 는 *test set* 으로 활용할 수 있다. (참고로 테스트셋과 트레이닝셋은 랜덤하게 분리하는 편이 좋다.)

Evaluating your hypothesis

Dataset:



(<http://blog.csdn.net/linuxcumt>)

linear regressoin 에 트레이닝 셋과 테스트셋을 분리하는 과정을 알아 보면

- (1) $\theta(\text{theta})$ 를 트레이닝 셋에 대해 학습시켜 트레이닝 에러를 최소화 하는 $J(\theta)$ 를 찾는다.
- (2) 학습된 $\theta(\text{theta})$ 에 대해 테스트 셋을 돌려 *test error* 를 찾는다.

그럼 *linear regression* 말고 *classification* 에는 어떻게 적용할까?

마찬가지로 $\theta(\text{theta})$ 에 대해 $J(\theta)$ 를 찾고, 여기에 $J_{\text{test}}(\theta)$ 를 돌려 테스트 에러를 찾는다.

아니면 아래 그림에서 볼 수 있듯이 *misclassification error* 를 이용해도 된다. 보면 알겠지만 같은 정의다. $y = 0$ 일때 $h(x) < 0.5$ 이어야 하고, $y = 1$ 일때 $h(x) \geq 0.5$ 이어야 하기 때문에 엇갈리게 나온 경우 *err* 함수에서 1 을 리턴해, 이 값을 모두 합한 뒤 전체 테스트 셋의 숫자로 나누면 테스트 에러값을 구할 수 있다.

Training/testing procedure for logistic regression

- - Learn parameter θ from training data
 - Compute test set error: m_{test}
 - $$J_{\text{test}}(\theta) = -\frac{1}{m_{\text{test}}} \sum_{i=1}^{m_{\text{test}}} y_{\text{test}}^{(i)} \log h_{\theta}(x_{\text{test}}^{(i)}) + (1 - y_{\text{test}}^{(i)}) \log h_{\theta}(x_{\text{test}}^{(i)})$$
 - Misclassification error (0/1 misclassification error):
- $$\text{err}(h_{\theta}(x), y) = \begin{cases} 1 & \text{if } h_{\theta}(x) \geq 0.5, y = 0 \\ & \text{or if } h_{\theta}(x) < 0.5, y = 1 \\ 0 & \text{otherwise} \end{cases} \text{ error}$$
- $$\text{Test error} = \frac{1}{m_{\text{test}}} \sum_{i=1}^{m_{\text{test}}} \text{err}(h_{\theta}(x_{\text{test}}^{(i)}), y_{\text{test}}^{(i)}).$$

(<http://blog.csdn.net/linuxcumt>)

Model Selection

당연한 이야기지만 *training set* 에 가설이 *well fit* 되어있기 때문에 트레이닝셋에 포함되지 않은 새로운 경향의 데이터를 만나면 에러가 많이 생길 수 있다.

Once parameters were fit to some set of data (training set), the error of the parameters as measured on that data (the training error $J(\theta)$) is likely to be lower than the actual generalization error.

d 를 *degree of polynomial* (가설의 다항식 차수) 이라 하자. 그럼

$d = 1, 2, \dots, 10$ 중에 어떤 걸 택하는 게 좋을까?

Model selection

$d = 1$ 1. $\rightarrow h_{\theta}(x) = \theta_0 + \theta_1 x \rightarrow \Theta^{(1)} \rightarrow J_{test}(\Theta^{(1)})$

$d = 2$ 2. $\rightarrow h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 \rightarrow \Theta^{(2)} \rightarrow J_{test}(\Theta^{(2)})$

$d = 3$ 3. $\rightarrow h_{\theta}(x) = \theta_0 + \theta_1 x + \dots + \theta_3 x^3 \rightarrow \Theta^{(3)} \rightarrow J_{test}(\Theta^{(3)})$

\vdots

$d = 10$ 10. $\rightarrow h_{\theta}(x) = \theta_0 + \theta_1 x + \dots + \theta_{10} x^{10} \rightarrow \Theta^{(10)} \rightarrow J_{test}(\Theta^{(10)})$

Choose $\theta_0 + \dots + \theta_5 x^5 \leftarrow$

How well does the model generalize? Report test set error $J_{test}(\theta^{(5)})$.

Problem: $J_{test}(\theta^{(5)})$ is likely to be an optimistic estimate of generalization error. I.e. our extra parameter ($d = \text{degree of polynomial}$) is fit to test set.

Handwritten notes: $d = \text{degree of polynomial}$ (circled in red), $\Theta^{(5)}$ (circled in red), $\Theta_0, \Theta_1, \dots$ (boxed in blue)

Andrew

(<http://blog.csdn.net/linuxcunt>)

각각에서 나오는 파라미터 벡터를 $\theta^{(1)}, \theta^{(2)}, \dots$ 이라 하자. 그리고 여기서 나오는 테스트 셋의 에러를 $J_{test}(\theta^{(1)})$, $J_{test}(\theta^{(2)})$ 등이라 하면 이 값을 모두 조사해 최소로 나오는 d 를 가진 모델을 택한다.

그러나 문제는 이렇게 선택한 모델이 *optimistic estimate of generalization error* 라는 점이다. 테스트 셋에 대해서 가장 적은 에러를 모델이 보여준다 해도 실제 데이터에 대해 똑같이 적은 에러를 보여주리라고 확신할 수 없다.

Train / Validation / Test Sets

이 문제를 해결하기 위해 *training set* 을 60%/20%/20% 로 나누어 각각을 *training set*, *cross validation set (CV)*, *test set* 이라 하자. 그리하여 각각의 에러를 구할 수 있다.

여기서 CV 에 대한 $error$ 가 최저인 모델을 택하면 이 모델은 $test\ set$ 에 대해서는 fit 되어 있지 않기 때문에 $test\ error$ 가 $estimate\ generalization\ error$ 라 볼 수 있다.

Train/validation/test error

Training error:

$$\rightarrow J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

Cross Validation error:

$$\rightarrow J_{cv}(\theta) = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} (h_{\theta}(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$$

Test error:

$$J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (h_{\theta}(x_{test}^{(i)}) - y_{test}^{(i)})^2$$

(<http://blog.csdn.net/linuxcumt>)

다시 정리해 보면 먼저 θ 를 $training\ set$ 에 대해 학습 시켜 θ 값을 얻은 뒤, $cross\ validation\ (CV)$ 에 대해 $error$ 를 구해 가장 작은 값을 갖는 모델을 고른다.

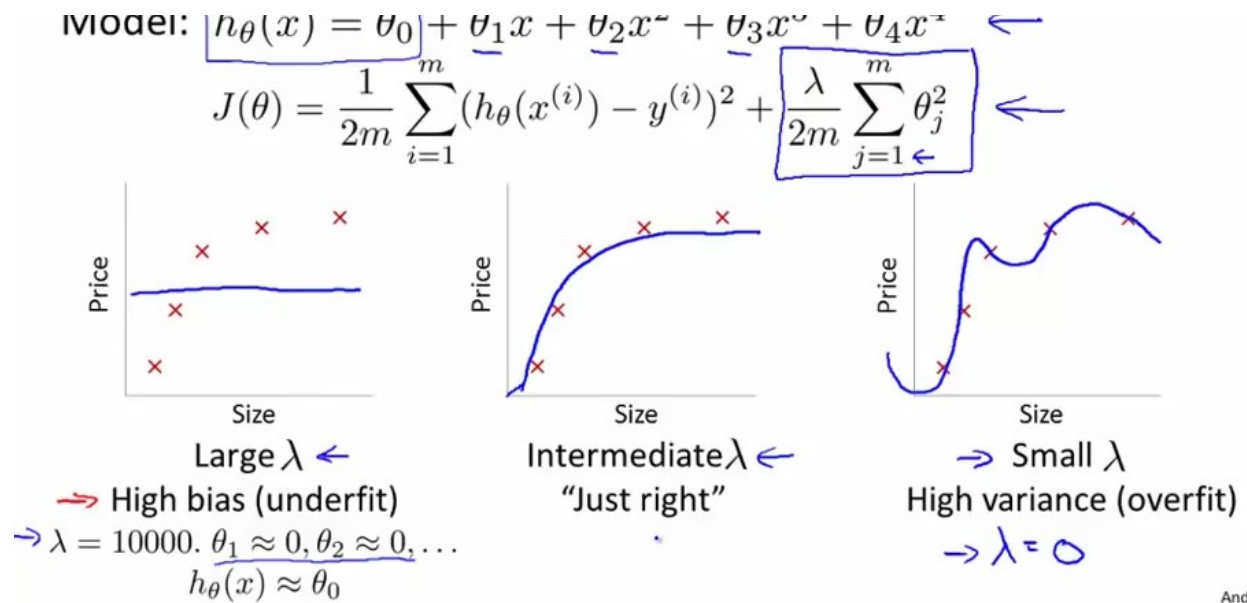
이제 이 모델에 대해서 $test\ error$ 를 구하면 이 모델은 테스트 셋에 대해서 fit 되지도, 가장 적은 에러를 가지는지 검사되지도 않은 데이터이므로 일반적인 에러값에 대한 추정치라 볼 수 있다.

일반적으로 $CV\ error$ 는 $test\ error$ 보다 더 작은 값을 가지는데, 이는 선택한 모델의 θ 가 $CV\ set$ 에 대해 최저치를 갖도록 fit 되어있기 때문이다.

Diagnosing Bias vs Variance

Linear regression with regularization

1. 2. 3. 4.



(<http://blog.csdn.net/linuxcunt>)

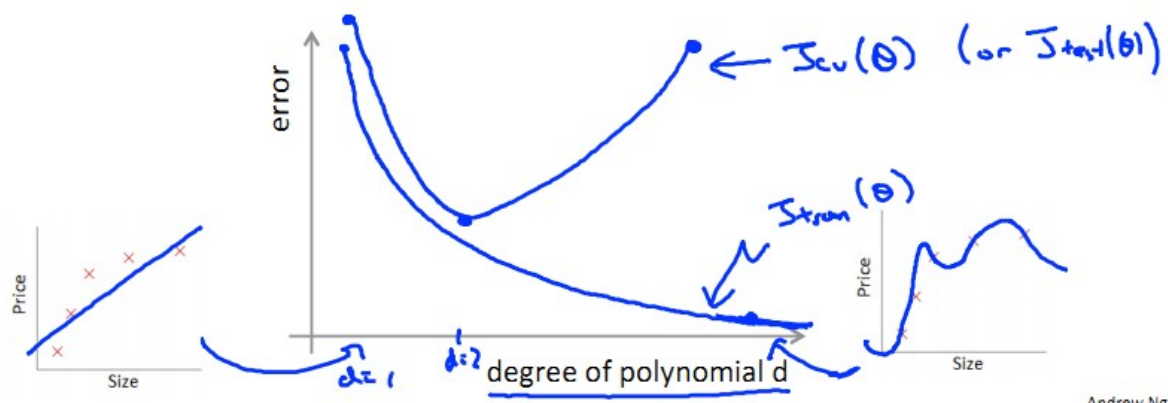
그림에서 볼 수 있듯이 $d=1$ 인 경우엔 *underfit*, $d=4$ 인 경우엔 *overfit* 이 발생한다. 다른말로 각각 *high bias*, *high variance* 라 부른다.

아래와 같이 가로 축을 d , 세로 축을 *error* 라 하면 d 가 증가할 수록 *training error* 는 0 에 가까워진다. 반면 *CV set* 에 대해서는 하나의 d 만 최저치를 가지고 나머지는 그 보다 높기 때문에 아래와 같은 그래프를 그릴 수 있다.

Bias/variance

Training error: $J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$

Cross validation error: $J_{cv}(\theta) = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} (h_{\theta}(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$ (or $J_{test}(\theta)$)



(<http://blog.csdn.net/abcjennifer>)

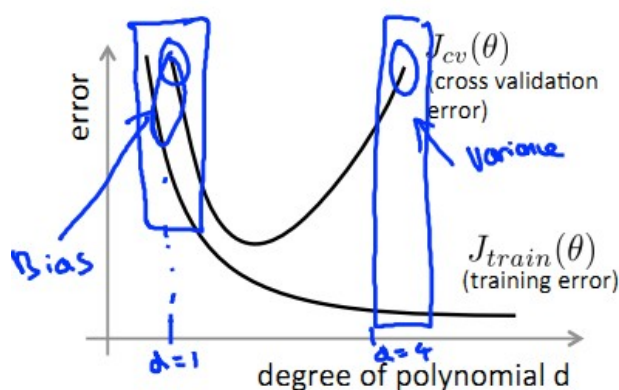
이 그래프가 시사하는 바는,

- (1) *underfit* 할 경우 d 가 작으므로 $J_{train}(\theta)$ 는 매우 크고, $J_{cv}(\theta)$ 는 거의 비슷한 값을 가지게 된다. (*bias problem*)
- (2) *overfit* 할 경우 d 가 크므로 $J_{train}(\theta)$ 는 매우 작고, $J_{cv}(\theta)$ 는 그보다 훨씬 크다. (*variance problem*)

따라서 $J_{train}(\theta)$ 값이 $J_{cv}(\theta)$ 과 얼마나 비슷한지 비교함으로써 *overfit* 혹은 *underfit* 되는지 판단할 수 있다.

Diagnosing bias vs. variance

Suppose your learning algorithm is performing less well than you were hoping. ($J_{cv}(\theta)$ or $J_{test}(\theta)$ is high.) Is it a bias problem or a variance problem?



Bias (underfit):
 $\rightarrow J_{train}(\theta)$ will be high
 $J_{cv}(\theta) \approx J_{train}(\theta)$

Variance (overfit):
 $\rightarrow J_{train}(\theta)$ will be low
 $J_{cv}(\theta) \gg J_{train}(\theta)$

(<http://blog.csdn.net/abcjennifer>)

여기 서 *bias vs variance* 의 이해를 위해 인용을 좀 하자면,

Bias, 즉 선입관이 크면, (좋게 말해서) 쫓대가 있고 (나쁘게 말해서) 고집이 세기 때문에 새로운 경험을 해도 거기에 크게 휘돌리지 않는다. 평소 믿음과 다른 결과가 관찰되더라도 한두 번 갖고는 콧방귀도 안 꺾며 생각의 일관성을 중시한다. (High Bias, Low Variance) 반대로 선입관이 작으면, (좋게 말하면) 사고가 유연하고 (나쁘게 말하면) 귀가 얇기 때문에 개별 경험이나 관찰 결과에 크게 의존한다. 새로운 사실이 발견되면 최대한 그걸 받아들이려고

하는 것이다. 그래서 어떤 경험을 했느냐에 따라서 최종 형태가 왔다갔다한다. (High Variance, Low Bias)

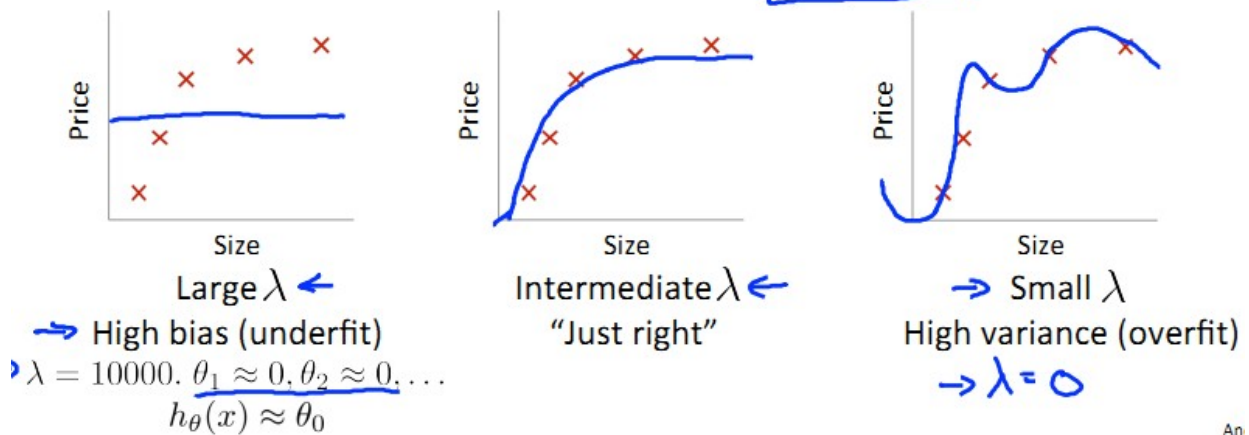
Regularization and Bias / Variance

regularization 이 끼어들면 λ 를 *bias vs variance* 문제에서 고려해야 한다. 아래 그림을 보자.

Linear regression with regularization

Model:
$$h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^m \theta_j^2$$



(<http://blog.csdn.net/abcjennifer>)

λ 가 크면 당연히 *high bias*, 매우 작으면 *high variance* 다. 그러면 중간 값을 찾아야 한다는건 알겠는데, 어느정도가 적당한 값일까?

Choosing the regularization parameter λ

$$h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^m \theta_j^2$$

$\rightarrow J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$ $J(\theta)$

$$J_{cv}(\theta) = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} (h_{\theta}(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$$

$$J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (h_{\theta}(x_{test}^{(i)}) - y_{test}^{(i)})^2$$

J_{train}
 J_{cv}
 J_{test}

(<http://blog.csdn.net/linuxcunt>)

이 전과 비교했을때 $J(\theta)$ 에 *regularization term* 이 추가되었지만 $J_{train}(\theta)$ 이
 나 $J_{cv}(\theta)$, $J_{test}(\theta)$ 에는 *regularization term* 이 없다는 점에 유의하자.

Choosing the regularization parameter λ

Model: $h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^m \theta_j^2$$

1. Try $\lambda = 0 \leftarrow \min J(\theta) \rightarrow \theta^{(1)} \rightarrow J_{cv}(\theta^{(1)})$
 2. Try $\lambda = 0.01 \rightarrow \min_{\theta} J(\theta) \rightarrow \theta^{(2)} \rightarrow J_{cv}(\theta^{(2)})$
 3. Try $\lambda = 0.02 \rightarrow \theta^{(3)} \rightarrow J_{cv}(\theta^{(3)})$
 4. Try $\lambda = 0.04$
 5. Try $\lambda = 0.08 \rightarrow \theta^{(5)} \rightarrow J_{cv}(\theta^{(5)})$
 - \vdots
 12. Try $\lambda = 10 \rightarrow \theta^{(12)} \rightarrow J_{cv}(\theta^{(12)})$
- \uparrow 10.24
 Pick (say) $\theta^{(5)}$. Test error: $J_{test}(\theta^{(5)})$

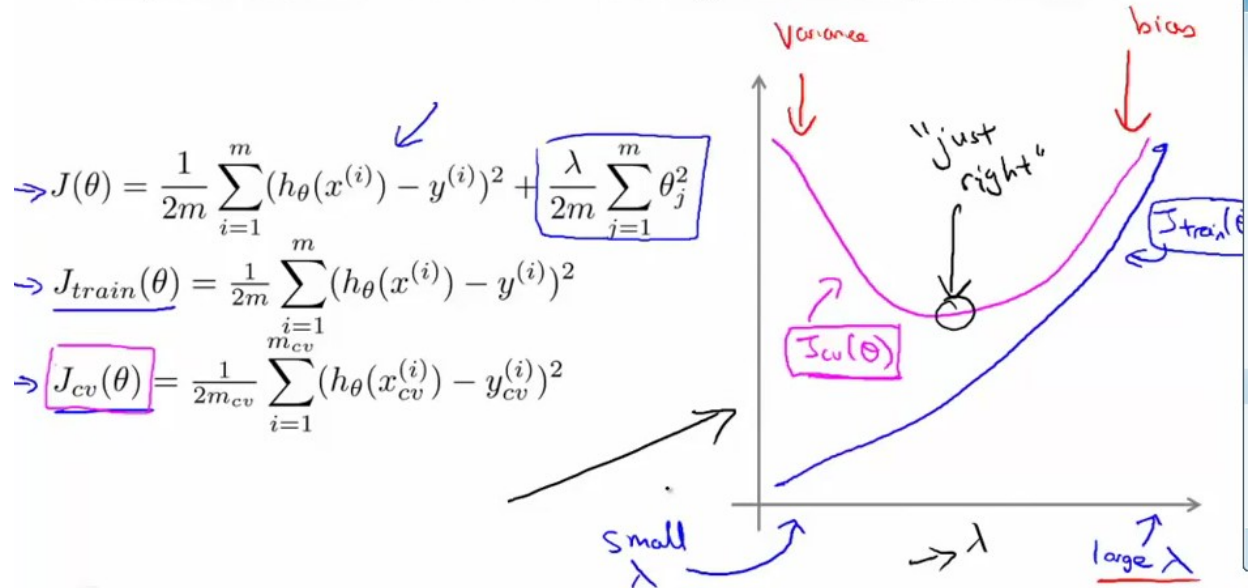
Andrew T

(<http://blog.csdn.net/linuxcunt>)

모델을 선택했다면 λ 를 천천히 증가시켜가면서 각각에 대해 $J(\theta)$ 를
 구한다. 그리고 이 값을 이용해 구한 $J_{cv}(\theta)$ 가 가장 적은 에러 값을 가지는
 λ 를 구하면 된다. *model selection* 과 비슷하다.

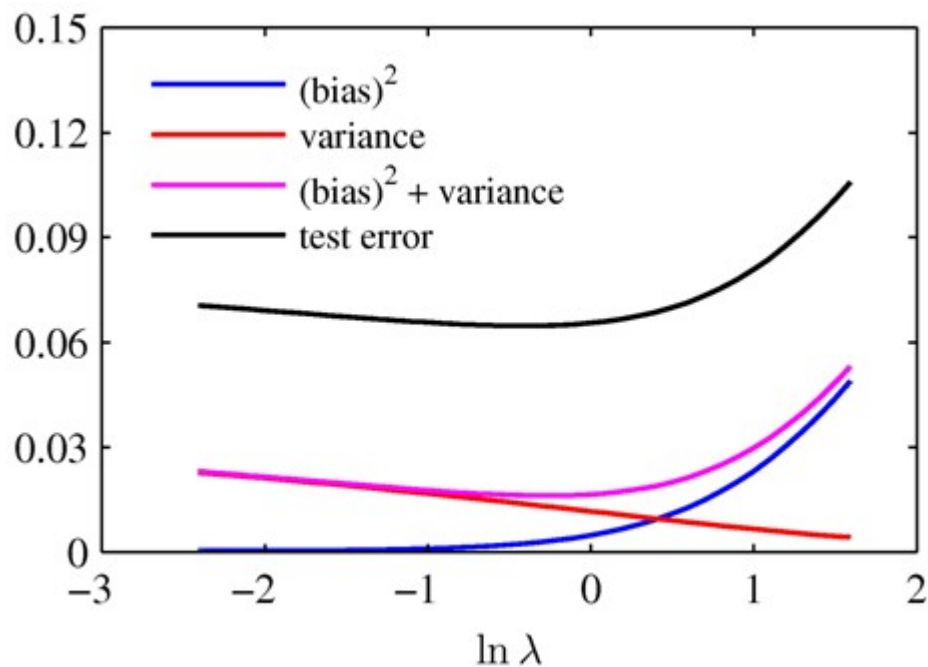
λ 과 *CV error*, *training error* 간 관계를 알아보자면 아래와 같다. 위에서 언급 했
 듯이 λ 가 크면 *bias*, 0 에 가까우면 *variance* 임을 확인할 수 있다.

Bias/variance as a function of the regularization parameter λ



(<http://blog.csdn.net/linuxcunt>)

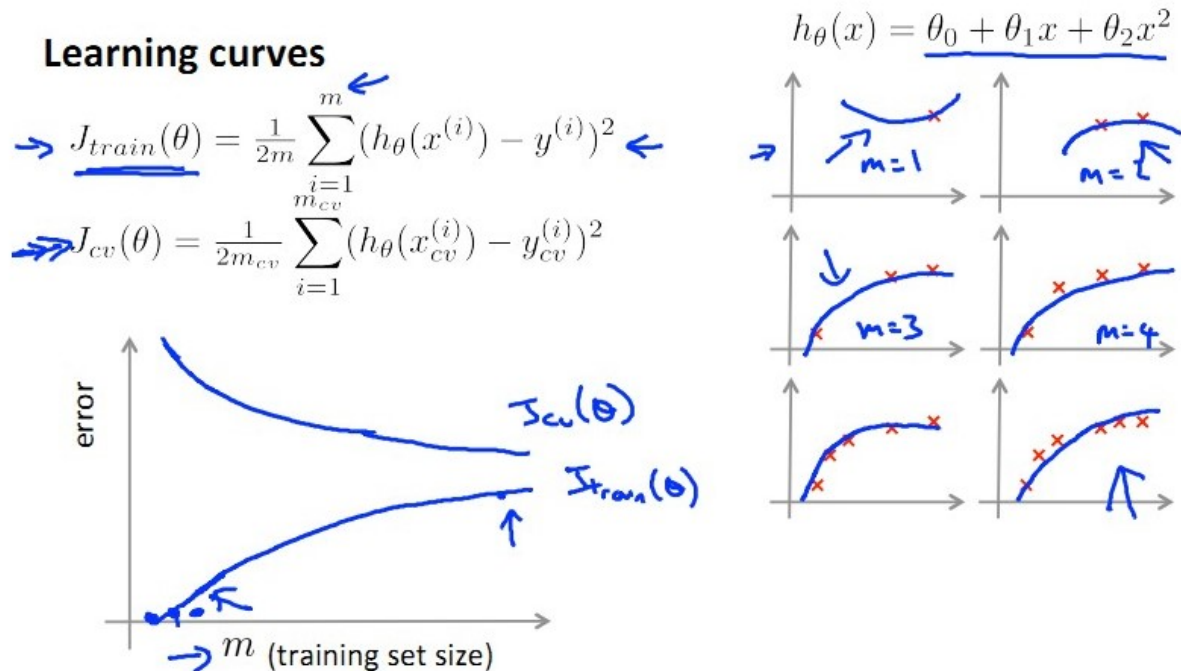
bias 와 *variance*, 그리고 *lambda* 의 관계는 아래 그래프에서도 확인할 수 있다.



(<http://blog.csdn.net/abcjennifer>)

Learning Curves

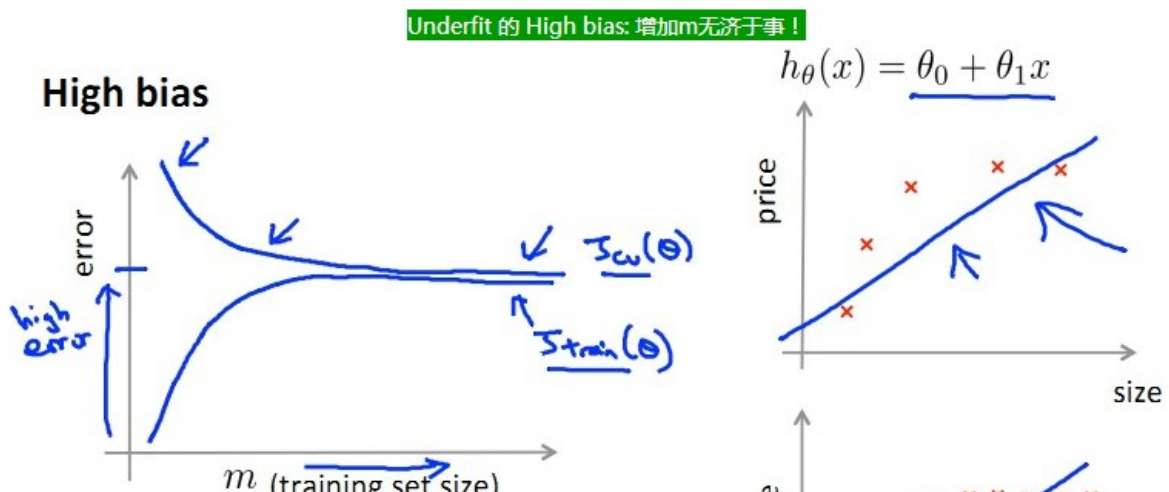
전체 트레이닝 셋의 사이즈 m 이 커질때 에러는 어떻게 되는가 그래프로 한번 보자.



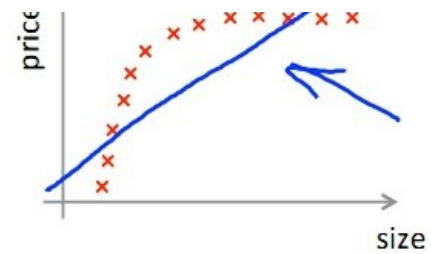
(<http://blog.csdn.net/linuxcunt>)

간단히 생각해 보면 m 의 사이즈가 클수록 *training set* 의 에러는 점점 늘어나고, m 이 커지면 커질수록 *generalize* 가 가능하므로 *CV error* 는 점점 줄어든다.

high bias 인 경우 처음엔 *training error* 이 매우 크다가, m 이 클수록 *training error* 의 증가율이 작아지므로



If a learning algorithm is suffering from high bias, getting more training data will not (by itself) help much.



(<http://blog.csdn.net/linuxcunt>)

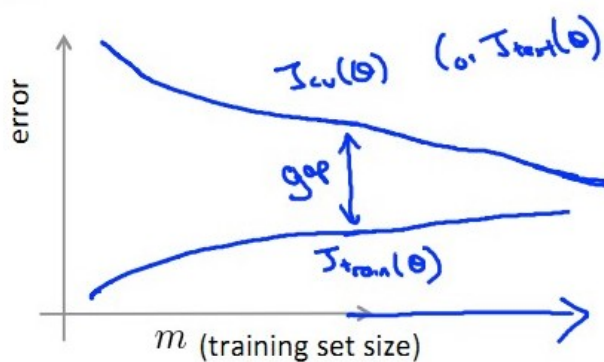
이 그림이 시사하는 바는

high bias 알고리즘이라면 m 이 을 많이 수집한다 해도 $J_{cv}(\theta)$ 의 감소율이 적기 때문에 별 도움이 되지 못한다. 다시 말해 m 을 많이 투입해도 얻어지는 *training error* 와 *CV error* 의 차이는 미미하다.

반면 *high variance* 의 경우에는 m 이 크면 클수록 *training error* 의 증가율이 점점 줄어들고, *overfit* 이기 때문에 *CV error* 는 *training set* 과 차이가 많이 난다. 그래프는

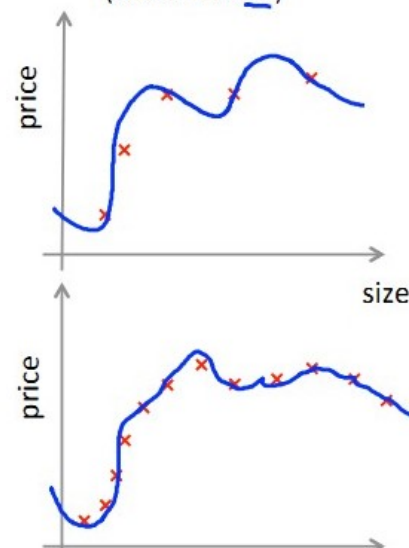
Overfit의 High Variance: 增加 m 使得 $J(\text{train})$ 和 $J(\text{cv})$ 之间gap减小, 有助于performance提高!

High variance



If a learning algorithm is suffering from high variance, getting more training data is likely to help. ←

$$h_{\theta}(x) = \theta_0 + \theta_1 x + \dots + \theta_{100} x^{100} \quad (\text{and small } \lambda)$$



(<http://blog.csdn.net/linuxcunt>)

결국

high variance 일 경우 m 을 많이 투입하면 할수록 낮은 *CV error* 를 얻는데 도움이 된다.

다시 말해 이 두가지 경우는 *training error* 와 *CV error* 의 차이가 꽤 클때는 m 을 높이면 낮은 *CV error* 를 적은 비용으로 얻을 수 있다는 뜻이다.

Applying to Neural Network

이제 처음에 나왔던 6가지 경우를 고려해 보자.

- (1) Get more training examples -> *fixing high variance*
- (2) Try smaller sets of features -> *fixing high variance*
- (3) Try getting additional features -> *fixing high bias*
- (4) Try adding polynomial features -> *fixing high bias*
- (5) Try decreasing λ -> *fixing high bias*
- (5) Try increasing λ -> *fixing high variance*

Debugging a learning algorithm:

Suppose you have implemented regularized linear regression to predict housing prices. However, when you test your hypothesis in a new set of houses, you find that it makes unacceptably large errors in its prediction. What should you try next?

- Get more training examples → *fixes high variance*
- Try smaller sets of features → *fixes high variance*
- Try getting additional features → *fixes high bias*
- Try adding polynomial features ($x_1^2, x_2^2, x_1x_2, \text{etc}$) → *fixes high bias*
- Try decreasing λ → *fixes high bias*
- Try increasing λ → *fixes high variance*

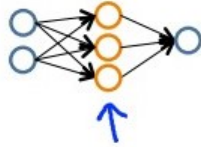
(<http://blog.csdn.net/linuxcunt>)

bias vs variance 문제를 *neural network* 에 적용시켜보자.

- (1) 작은 사이즈의 신경망이라면 계산 비용은 저렴한 대신 *underfit* 할 수 있다.
- (2) 큰 사이즈의 신경망이라면 계산 비용은 비싸고 *overfit* 할 수 있다. 따라서 *regularization* 을 이용해 *overfit* 되는 정도를 줄일 수 있다.

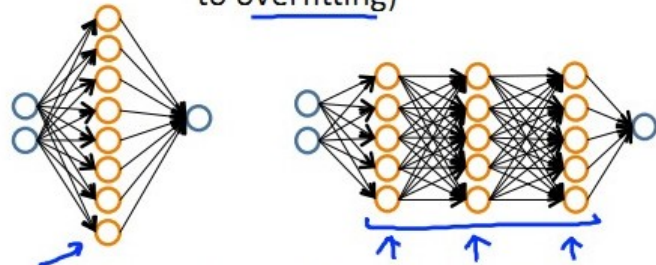
Neural networks and overfitting

→ “Small” neural network
(fewer parameters; more prone to underfitting)



Computationally cheaper

→ “Large” neural network
(more parameters; more prone to overfitting)



Computationally more expensive.

Use regularization (λ) to address overfitting.

$$J_{\lambda}(w)$$

(<http://blog.csdn.net/linuxcumt>)

Machine Learning System Design

간단한 스팸 분류기를 작성한다고 하자. *supervised learning* 을 위해서

- (1) x = features of email (*choose 100 words indicative of spam or not*)
- (2) y = spam 1 or not spam 0

Building a spam classifier

Supervised learning. x = features of email. y = spam (1) or not spam (0).

Features x : Choose 100 words indicative of spam/not spam.

E.g. deal, buy, discount, andrew, now, ...

$$x = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ \vdots \\ 1 \\ \vdots \end{bmatrix} \begin{matrix} \text{andrew} \\ \text{buy} \\ \text{deal} \\ \text{discount} \\ \vdots \\ \text{now} \\ \vdots \end{matrix} \quad x \in \mathbb{R}^{100}$$

$$x_j = \begin{cases} 1 & \text{if word } j \text{ appears in email} \\ 0 & \text{otherwise} \end{cases}$$

From: cheapsales@buystufffromme.com
To: ang@cs.stanford.edu
Subject: Buy now!

Deal of the week! Buy now!

Note: In practice, take most frequently occurring n words (10,000 to 50,000) in training set, rather than manually pick 100 words.

(<http://blog.csdn.net/linuxcunt>)

각 단어가 이메일 본문에 등장했는지 아닌지를 각 *feature* 의 값으로 사용한다. (1 or 0)

일반적으로는 100개를 수동으로 고르는게 아니라, 스팸에서 많이 사용된 단어를 n 개 골라 사용한다.

그럼 *low error* 를 얻기 위해서는 무엇을 해야할까?

- (1) Collect lots of data : 항상 도움이 되진 않는다.
- (2) Develop sophisticated features based on email routing information
- (3) Develop sophisticated features for message body. e.g should “discount” and “discounts” be treated as the same word?
- (4) Develop sophisticated algorithm to detect misspelings e.g m0rtgage

등등 의 다양한 방법을 고안할 수 있다. 이 중 무엇을 선택해야 할까? 좀 더 체계적인 방법은 없을까? 여기 몇 가지 가이드라인이 있다.

1. Start with a simple algorithm that can implement quickly. Implement it and test it on your corss-validation data.
2. Plot learning curves to decide if more data, more features, etc. are likely to help.
3. Error analysis: manually examine the examples (in corss validation set) that your algorithm made errors on. See if you spot any systematic trend in what type of examples it is making errors on.

Error Analysis

error analysis 하는 방법은 *CV error* 를 발견했을 때, 각각의 에러를 수동으로 검사하면서 분류하는 것이다.

이메일의 타입이 무엇인지, 혹은 어떤 *feature* 가 알고리즘에서 이 이메일을 분류하는데 도움이 될만한지 생각해 본다.

Error Analysis

$m_{CV} = 500$ examples in cross validation set

Algorithm misclassifies 100 emails.

Manually examine the 100 errors. and categorize them based on:

- (i) What type of email it is *pharma, replica, steal passwords, ...*
- (ii) What cues (features) you think would have helped the algorithm classify them correctly.

Pharma: *12* → Deliberate misspellings: *5*
 Replica/fake: *4* (m0rgage, med1cine, etc.)
 Steal passwords: *53* → Unusual email routing: *16*
 Other: *31* → Unusual (spamming) punctuation: *32*

(<http://blog.csdn.net/linuxcumt>)

error analysis 가 에러가 나타난 이유에 대한 어떤 경향을 제공할 수 있기 때문에 간단히 먼저 구현해 보고 분석 해 보는것도 나쁘지 않다.

error analysis 는 실제로 분석 결과를 새로운 알고리즘에 적용했을때 *performace* 가 더 좋을지 알려주지 않는다. 따라서 해보고 *numerical evaluation* 을 비교해 본다.

The importance of numerical evaluation

Should discount/discounts/discounted/discounting be treated as the same word?

Can use “stemming” software (E.g. “Porter stemmer”)

universe/university.

Error analysis may not be helpful for deciding if this is likely to improve performance. Only solution is to try it and see if it works.

Need numerical evaluation (e.g., cross validation error) of algorithm’s performance with and without stemming.

Without stemming: *5% error* With stemming: *3% error*

Distinguish upper vs. lower case (Mom/mom): *3.2%*

(<http://blog.csdn.net/linuxcumt>)

Skewed Classes

암을 진단한다고 하자. *logistic regression* 을 구현했고, 놀랍게도 *test error* 가 1% 라고 하자.

근데, 만약에 환자중에 0.5% 만 암환자라면, 차라리 모두 암이 아니라고 진단하는 다음의 함수가 더 에러가 낮다.

```
function y = predictCancer(x)
    y = 0;
    return
```

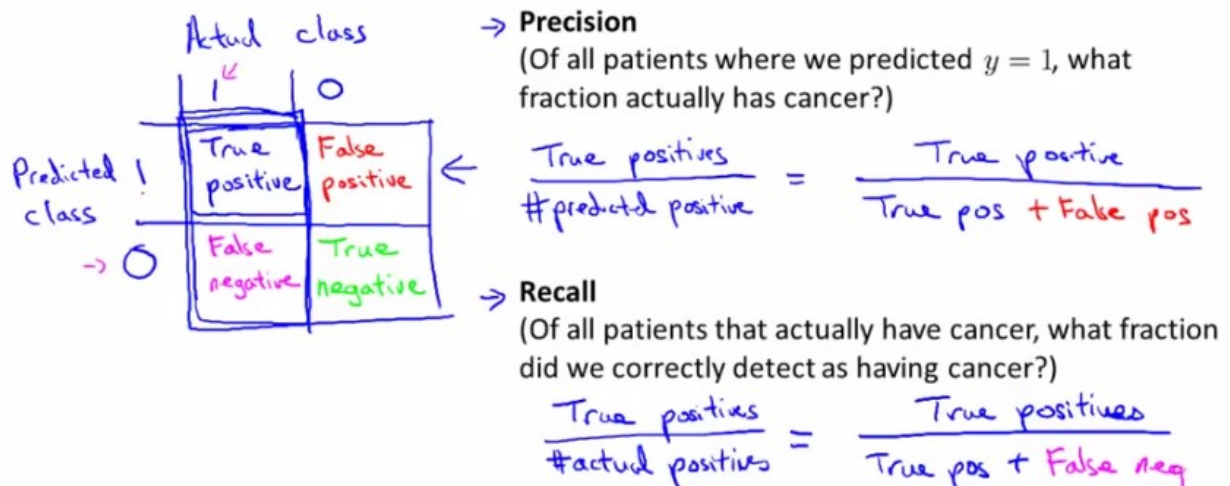
이렇게 확률이 희박한 *class* 를 **skewed class** 라 부른다. 또 한가지 사실을 알 수 있는데, *error* 가 낮다고 해서 항상 좋은 알고리즘이 아니라는 사실이다. $y = 0$ 은 99.5%의 정확도를 보여주지만 알고리즘이 아니다. 에러값 말고 다른 평가방법이 필요하다!

Precision / Recall

그림을 먼저 보자. 예측 여부와 실제 값에 따라서 2×2 매트릭스를 붙일 수 있다.

Precision/Recall

$y = 1$ in presence of rare class that we want to detect



(<http://blog.csdn.net/abcjennifer>)

여기서 *precision* 과 *recall* 이란 개념을 끌어낼 수 있는데

****Precision: **** Of All patients where we predicted $y = 1$, what fraction actually has cancer?

****Recall: **** Of all patients that actually have cancer, what fraction did we correctly detect as having cancer?

다시 말해 *precision* 은 우리가 암이 있다고 진단한 환자중 실제 암이 있는 환자의 비율이고, *recall* 은 실제 암이 있는 환자 중 우리가 암이 있다고 진단한 환자의 비율이다.

위의 함수처럼 $y = 0$ 으로 진단하는 경우 *true positive* = 0 이므로 *recall* = 0 이다.

단순히 *error* 만으로 판단하는 것은 위의 예처럼 잘못된 판단일 수 있다. 따라서 *skewed class* 가 있더라도 *precision* 과 *recall* 을 보면 알고리즘에 속임수가 있는지, 없는지를 파악할 수 있다.

Trading off Precision and Recall

Logistic regression: $0 \leq h_{\theta}(x) \leq 1$

Predict 1 if $h_{\theta}(x) \geq 0.5$

Predict 0 if $h_{\theta}(x) < 0.5$

$$\text{precision} = \frac{\text{true positives}}{\text{no. of predicted positive}}$$

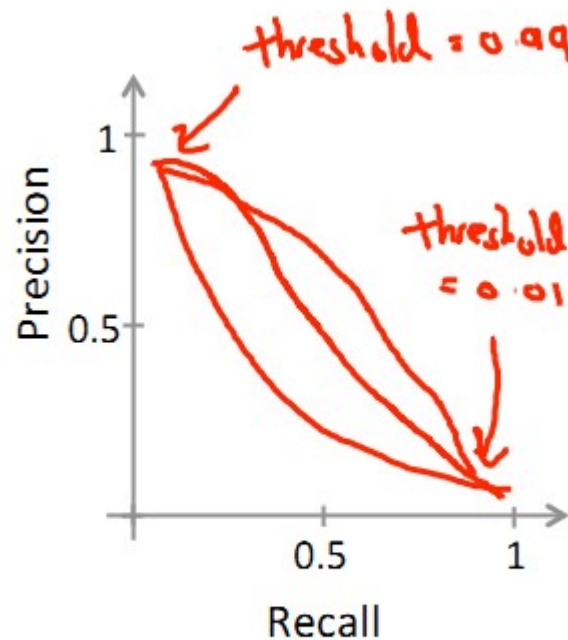
$$\text{recall} = \frac{\text{true positives}}{\text{no. of actual positive}}$$

(<http://blog.csdn.net/linuxcunt>)

일반적으로 $h(x) \geq 0.5$ 일 경우에 1 을, $h(x) < 0.5$ 일 경우에 0 을 예측하는데, 이 수치를 좀 더 올려 0.7 이상 또는 미만으로 예측한다 해 보자.

이 경우 좀 더 확실한 환자만 암이라 진단하므로 *precision* 은 올라가는 반면 *recall* 은 내려간다.

거꾸로 수치를 0.3 으로 낮추면 덜 확실해도 그냥 암이라 우기므로 *recall* 은 높아지겠지만 예측한 것중 실제 환자를 의미하는 *precision* 값은 떨어진다.



thres高对应高precision低recall ;

thres低对应低precision高recall ;

(<http://blog.csdn.net/linuxcumt>)

위 그림을 보면 *threshold* 에 따라서 *recall* 과 *precision* 값이 얼추 반비례하는 걸 볼 수 있다. 디테일에 따라서 구체적인 그래프의 모양은 다를 수 있다.

그럼 이제 문제는, *threshold* 를 고를 수 있느냐, 다시 말해 어느 (*precision*, *recall*) 쌍이 더 좋은가 하는 문제다.

	Precision(P)	Recall (R)
Algorithm 1	0.5	0.4
Algorithm 2	0.7	0.1
Algorithm 3	0.02	1.0

(<http://blog.csdn.net/linuxcunt>)

단순히 평균을 쓰면 $y = 1$ 로 예측하는 것과 같은 알고리즘들이 높은 값을 얻을 수 있다. 예를 들어 위 그림에서 *algorithm 3* 가 그렇듯이.

따라서 단순히 평균을 하기 보다는 *F1 score* 를 많이 쓴다.

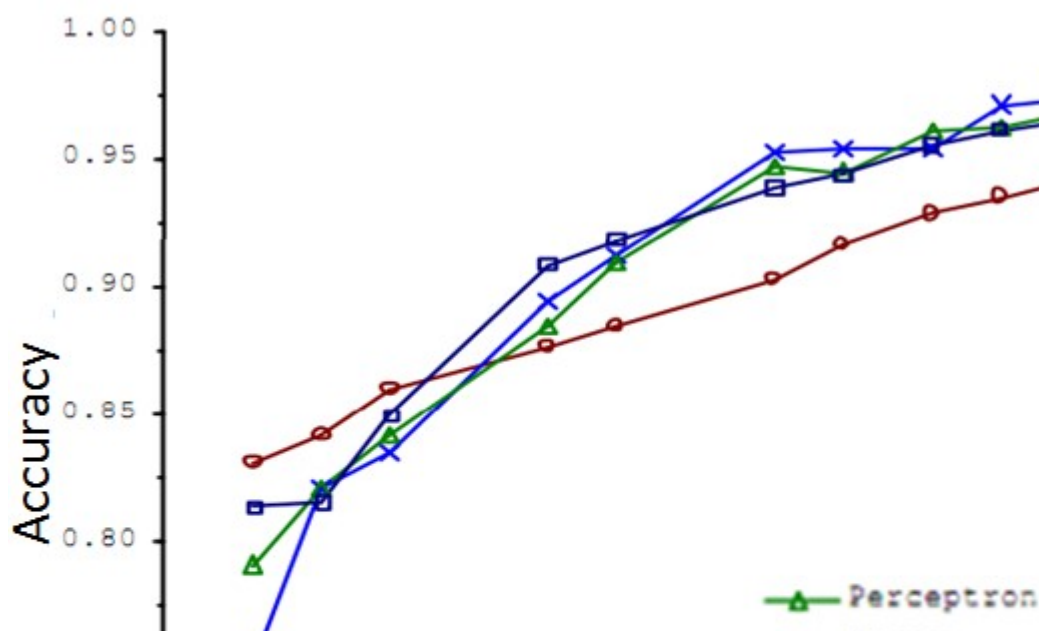
$$F_1 \text{ Score: } 2 \frac{PR}{P+R}$$

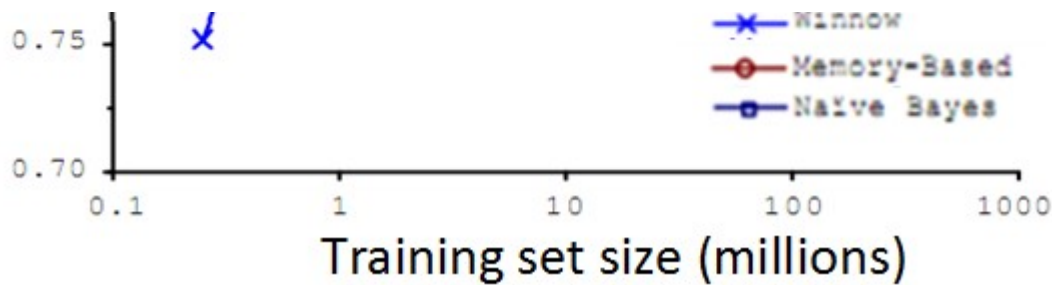
(<http://blog.csdn.net/linuxcunt>)

따라서 *CV set* 에 대해 높은 *F1 score* 를 가지는 *threshold* 를 택함으로써 좋은 알고리즘을 고를 수 있다.

Data for Machine Learning

지금까지는 *evaluation* 에 대한 논의였고, *data* 에 대한 이야기를 좀 더 해 보자. 앞에서는 단순히 데이터가 많다고 해서 좋다는 뉘앙스로 이야기를 했지만 실제 특정 상황에서, 특정 알고리즘은 다량의 데이터를 이용하면 좋은 성능을 내기도 한다. 실제 그런가 보자. 4개의 서로 다른 알고리즘을 트레이닝 셋 사이즈를 늘려가며 정확도를 비교한 결과다.





(<http://blog.csdn.net/linuxcumt>)

"It's not who has the best algorithm that wins, It's who has the most data."

항상 그렇지는 않다. 집 값을 예측 할 때 *feature* 로 사이즈 하나만 준다면 정확하게 예측하기란 불가능하다. 양이 문제가 아니고 집 값을 예측하기에 충분한 정보가 필요하다.

Large data rationale

- Use a learning algorithm with many parameters (e.g. logistic regression/linear regression with many features; neural network with many hidden units). low bias algorithms. ←

→ $J_{\text{train}}(\theta)$ will be small.

Use a very large training set (unlikely to overfit) low variance ←

→ $J_{\text{train}}(\theta) \approx J_{\text{test}}(\theta)$

→ $J_{\text{test}}(\theta)$ will be small

(<http://blog.csdn.net/linuxcumt>)

많은 수의 *parameter* 가 있다. 하자. *low bias* 기 때문에 $J_{\text{train}}(\theta)$ 는 작을 것이다. (*not underfit*)

그리고 여기에 *parameter* 보다 훨씬 많은 *training set* 을 사용한다면, *overfit* 하지 않는다 볼 수 있다. 따라서 *underfit* 도 아니고 *overfit* 도 아니므로

$J_{\text{test}}(\theta)$ 는 $J_{\text{train}}(\theta)$ 에 근사한 값을 가진다 볼 수 있다. 결국 작은 $J_{\text{test}}(\theta)$ 을 얻을 수 있다.

정리하자면, 충분한 양의 정보를 가지고 있고 (*large parameters*), 큰 사이즈의 데이터를 대상으로 알고리즘을 훈련 시킨다면 상당히 좋은 성능을 뽑아낼 수 있다는 훈훈한 이야기. (거꾸로 말하면, 반복하지만, 데이터만 많다고, 혹은 파라미터만 많다고 좋은 결과를 얻을 수 없다는 이야기)

References

- (1) *Machine Learning* by **Andrew NG**
- (2) <http://blog.csdn.net/linuxcumt>
- (3) <http://blog.csdn.net/abcjennifer>
- (4) <http://www.4four.us>

0 Comments lambda

 Login ▾ Recommend  Share

Sort by Best ▾



Start the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS 

Be the first to comment.

ALSO ON LAMBDA

하스켈로 배우는 함수형 언어 4

1 comment • a year ago

Junmo Roger Kang — 아이고 공부가 드디어 모나드에서 걸리네요 ㅌㅌ**하스켈로 배우는 함수형 언어 8**




1 comment • a year ago

NaDDu — 에라토스테네스의 체 알고리즘에 오류가 있는 것 같습니다. 15는 소수 아닙니다. 제가 하스켈을 공부한지 아직 일주**Functional Programming in Scala, Chapter 1**

1 comment • 4 years ago

Woojin Joe — 우와 정리 정말 잘하셨네요.. progfun... 정말 듣다 그만두다를 여러 번 반복하다 이제 5주차까지 들었는데, 올**Easy Scalaz 1**

1 comment • a year ago

Daesap — 좋은 포스트 감사합니다 >_<; 타입 클래스를 이해하는데 큰 도움이 되었습니다. leftMap(err => rollback; err); 는 오타 Subscribe  Add Disqus to your site Add Disqus Add  Privacy